Tutorial 1: Processes
Solutions

Vaishaal Shankar, Erik Krogen,
Dmitry Shkatov, Craig Bester

July 21, 2016

Contents

1 Questions

1.1 Hello World

What is the output produced by the following code? Assume the child's PID is 90210.
If you're not sure, try executing it on your machine. (Hint: There is more than one correct answer.)

```c
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>

int main( void ) {
    pid_t pid = fork();
    printf("Hello World: %d\n", pid);
}
```

Answer:

Hello World: 90210
Hello World: 0

OR

Hello World: 0
Hello World: 90210

Either process could be scheduled to print first

1.2 Forks

How many processes are created by the following code?
(Including the process created when you execute the program.)

```
int main( void ) {
    for (int i = 0; i < 3; i++) {
        pid_t pid = fork();
    }
}
```

Answer:

8 processes.
Each iteration of the loop doubles the number of processes. The children will continue the loop with the current value of i from the parent process; depending on the iteration they were spawned. Since the loop executes 3 times, 7 processes are spawned. Including the inital process brings the total number to 8.

1.3 Fork Bomb

Why is the following code bad for your system? (Don't try executing this code!)

```
int main( void ) {
    while( true )
        fork();
}
```

Answer:

The infinite loop will continually spawn new processes until the computer freezes. Each new process will also replicate, so the number of processes grows exponentially. Each process uses space in the process table and the operating system tries to context switch between all of them, making the computer unresponsive.

1.4 Stack Allocation

What is the output of the following code?

```c
int main( void ) {
    int stuff = 7;
    pid_t pid = fork();
    printf("Stuff is %d\n", stuff);
    if (pid == 0) {
        stuff = 6;
        printf("Stuff is %d\n", stuff);
    }

}
```

Answer:

Stuff is 7
Stuff is 7
Stuff is 6
OR
Stuff is 7
Stuff is 6
Stuff is 7

The entire address space is copied to the child process, so the stack allocated variable ‘stuff’ is initially 7 in the child, then changed to 6. The value in the parent process is not changed by the child. The second output is theoretically possible if the child process completes before the parent process is scheduled after the fork.

1.5 Heap Allocation

What is the output of the following code?

```c
int main( void ) {
    int *stuff = malloc( sizeof(int)*1 );
    *stuff = 7;
    pid_t pid = fork();
    printf("Stuff is %d\n", *stuff);
    if (pid == 0) {
        *stuff = 6;
        printf("Stuff is %d\n", *stuff);
    }
}
```

Answer:

Stuff is 7
Stuff is 7
Stuff is 6
OR
Stuff is 7
Stuff is 6
Stuff is 7

The entire address space is copied to the child process, so the heap allocated variable 'stuff' is initially 7 in the child, then changed to 6. The value in the parent process is not changed by the child. The second output is theoretically possible if the child process completes before the parent process is scheduled after the fork.

1.6 Simple Wait

```c
int main( void ) {
    pid_t pid = fork();
    int exit;
    if (pid != 0) {
        wait(&exit);
    }
    printf("Hello World: %d\n", pid);
}
```

1. What is the output of the code? Assume the child PID is 90210.

Answer:
Hello World: 0
Hello World: 90210

The order of printing is guaranteed because the parent waits for a child to complete before continuing.

2. Rewrite the code using the waitpid function instead of wait.

Answer:

```c
int main( void ) {
    pid_t pid = fork();
    int exit;
    if (pid != 0) {
        waitpid(pid,&exit,0);
    }
    printf("Hello World: %d\n", pid);
}
```

1.7 Nontrivial Wait

1. What is the exit code of the following program? Try to recognise a pattern. (You can view the exit code of the last command in bash with "echo $?")

   Answer:
   55. The foo function calculates Fibonacci numbers.

2. Why does this program start exhibiting odd behaviour when $n > 13$?
   (Hint: What is the datatype of the exit code?)

   Answer:
   Exit codes are only 1 byte. The 14^{th} and onward Fibonacci numbers are all greater than 255, so the numbers overflow.

3. What is the exit code for foo(14)?

   Answer:
   121

```
int foo( int n ) {
    if( n < 2 ) {
        exit( n );
    } else {
        int v1;
        int v2;

        pid_t pid = fork();
        if (pid == 0)
            foo( n - 1 );

        pid_t pid2 = fork();
        if( pid2 == 0 )
            foo(n - 2);

        waitpid( pid, &v1 ,0 );
        waitpid( pid2, &v2, 0);
        exit( WEXITSTATUS(v1) + WEXITSTATUS(v2) );
    }
}

int main( void ) {
    foo(10);
}
```

1.8 Exec

What is the output produced by the following code?

```
int main( void ) {
    char** argv = (char**) malloc(3*sizeof(char*));
    argv[0] = "/bin/ls";
    argv[1] = ".";
    argv[2] = NULL;
    for (int i = 0; i < 10; i++){
        printf("%d\n", i);
        if (i == 3)
            execv("/bin/ls", argv);
    }
}
```

Answer:
0
1
2
3
< *output of ls* >

1.9 Exec + Fork

Modify the above code so it prints both the output of ls and all the numbers from 0 to 9 (in an arbitrary order). You may not remove any lines from the original program; you can only add statements. (Hint: use fork()).

Answer:

```
int main( void ) {
    char** argv = (char**) malloc(3*sizeof(char*));
    argv[0] = "/bin/ls";
    argv[1] = ".";
    argv[2] = NULL;
    for (int i = 0; i < 10; i++){
        printf("%d\n", i);
        if (i == 3) {
            pid_t pid = fork();
            if(pid == 0)
                execv("/bin/ls", argv);
        }
    }
}
```